

# UNIVERSAL PERIPHERAL DEVICE CONTROLLER

## Field of the Invention

The present invention relates to sending information from a portable device to a  
5 remotely-located computer. More specifically, it relates to data transmission from a computer  
peripheral device over the Internet to a web server.

## Background

Most device communication systems require that custom software, drivers, and/or  
10 user interfacing software be installed on a personal computer (PC) in order to allow a  
peripheral device, such as a Palm Pilot, digital pad, or other peripheral device to  
communicate with the PC. In order for the peripheral device to transfer data to a remotely-  
located computer, such as a server, the peripheral device must first transfer the data to a local  
PC or other computer that has had the required custom software, drivers, and/or user  
15 interfacing software installed.

The necessity of installing customized software, drivers and user interface software  
onto a PC to enable a peripheral device to communicate with that PC or remotely-located PCs  
or servers creates problems. Excess memory is used, interference with other software can  
occur, upgrades need to be installed on the PC, the software has to be maintained, conflicts  
20 between drivers can develop, as well as conflicts between COM ports and other conflicts  
between the software required to run the peripheral device and software that runs on the PC  
for other purposes.

Moreover, whenever a user desires to transfer information from a peripheral device to  
a remotely-located computer, the user must either find a PC that has had the appropriate  
25 software installed or bring the software along so that he can install it on the nearest PC. What  
is needed is a method and system for transferring data from a peripheral device to a remote  
computer that is independent of what software is installed on a local PC or other device used  
to link the peripheral device to the Internet.

## Summary

A preferred embodiment of the present invention comprises a software plug-in that allows a peripheral device to communicate via a host (e.g., a personal computer) with a remote server through a communications port. The plug-in is preferably downloaded from the remote server and allows the peripheral device to communicate with that server or other remote servers, and the remote servers are able to communicate and control the peripheral device without any additional software being installed to the host.

## Brief Description of the Drawings

**FIG. 1** depicts two potential configurations of a preferred system.

**FIG. 2** is a flowchart showing preferred functionality of software of a preferred embodiment of the present invention.

## Detailed Description of Preferred Embodiments

**FIG. 1** depicts two potential configurations of a preferred system. In a preferred embodiment, a user attaches a peripheral device, such as a digital camera **50** or a digitizer pad **60**, to a communications port (whether on a PC **10**, a Web Phone, an Internet-enabled Palm Pilot **30** or another Internet access device) and then uses a web browser to access a system web server **40**.

Upon connecting to the system web server **40**, the user downloads a plug-in to the PC **10**, for example, that allows the peripheral device to communicate to the remote server **40** through the communications port. Herein, the terms "plug-in" and "plug-in computer program" include software such as a browser plug-in, a PRC (also known as a "Palm Resource" or "Palm Application"), or an ActiveX Control.

The plug-in allows the peripheral device to communicate with remote servers of the system and the remote servers are able to communicate with and control the peripheral device. Source code for a browser plug-in written in the C++ programming language and that uses the Netscape Plug-in Application Programming Interface (API) for running on Windows platforms is included in the Appendix at the end of this description.

**FIG. 2** is a flowchart showing preferred functionality of the plug-in and steps of a preferred method. A *host* is a device (PC with browser **10**, Internet-enabled Palm device **30**,

or other Internet-enabled device) that an *input device* (peripheral device – e.g., digitizer pad 60, digital camera 50, non-Internet-enabled Palm Pilot) is connected to via a communications port of the host. As used herein, the term “communications port” includes an RS-232 serial port, a USB port, an infrared port, or a Bluetooth port. Thus, the term “input device” does not include a keyboard or a mouse. In the following description, the actions of the host are controlled by a plug-in that has preferably been downloaded over the Internet. At step 105 a host watches for data from an input device. At step 110 the host checks whether a request from the input device to upload data has been detected. If not, the host continues at step 105 to watch for data from the input device.

If at step 110 a request from the input device to upload data has been detected, then at step 115 the host initiates an upload process, and at step 120 data is transferred from the input device to the host’s data storage. The data transfer is performed using the input device’s specific communications protocol. This protocol is utilized by the plug-in. In a preferred embodiment, a different plug-in is used for each different communications protocol. In an alternate embodiment, a single plug-in comprises software to enable communications with a plurality of devices that use a plurality of different communications protocols.

At step 125, the host checks whether the data transfer is complete. If not, then step 120 is repeated and/or continued, as appropriate. If at step 125 data transfer is complete, then at step 130 the host prepares the received and stored data for transmission to a system web server 40. The data may be reformatted at this step. Preferably, it is packaged into a standard HTML POST command data packet.

At step 135, the host initiates transmission of the received and stored data to a system web server 40. At step 140 the data is transferred from the host to the web server 40 through a browser installed on the host and the web server 40. The data is transferred to the system web server 40 using an API provided by the browser.

At step 145 the host checks whether the data transfer to the web server 40 is complete. If not, then step 140 is continued or repeated, as appropriate. If at step 145 the data transfer is complete, then at step 150 the host reports the status of the data transmission to the user (success or failure). At step 155 the host returns to a monitoring state and repeats step 105.

Although the present invention has been described with respect to input devices such as digitizer pads and digital cameras, and Internet-enabled devices such as PCs with browsers

and Internet-enabled Palm Pilots or other personal digital assistants (PDAs), those skilled in the art will recognize that the invention may be used to transmit data from any input device to a web server, if the input device is configured to transmit data to a PC or other device that can be connected to the Internet.

5

## Appendix

Source code for a browser plug-in written in the C++ programming language and that uses the Netscape Plug-in Application Programming Interface (API) for running on Windows platforms:

```
10 //=====
#include <stdio.h>
#include <string.h>
#include "npapi.h"
#include <windows.h>
15 #include "resource.h"

#pragma comment(lib, "Wsock32.lib")

#import "C:\dev\vc\timbre1_plugin\Windows\InkXfer.tlb"
20 using namespace INKXFERLib;

LRESULT CALLBACK PluginWindowProc( HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam);
const char* gInstanceLookupString = "instance->pdata";
25
HANDLE      hComm;
int         gConnected = 0;
static unsigned char *inBuffer=NULL;
static unsigned char *outBuffer=NULL;
30 DWORD      inBufferSize;
DWORD      outBufferSize;

#define kMAX_STRS 25
```

```
char    gMessageTextArray[kMAX_STRS][256]; // = {"Line 1","Line 2","Line
3","Line 4","Line 5","Line 6","Line 7","Line 8","Line 9"};
```

```
int      gMessageTextIndex = 0;
```

```
int      gNumLines=kMAX_STRS;
```

5

```
typedef struct _PluginInstance
```

```
{
```

```
    NPWindow*    fWindow;
```

```
    uint16       fMode;
```

10

```
    HWND         fhWnd;
```

```
    WNDPROC      fDefaultWindowProc;
```

```
    NPP          gInstance;
```

15

```
    char         gHostName[256];
```

```
    char         gHostPort[8];
```

```
    char         gUID[8];
```

```
    char         gProxyName[256];
```

```
    char         gProxyPort[8];
```

20

```
    char         gComPort[8];
```

```
    char         gComSpeed[10];
```

```
    char         gSourceURL[256];
```

```
    BOOL         gVerbose;
```

```
    char         gVersion[6];
```

25

```
    BOOL         bTransNote;
```

```
    BOOL         gReading;
```

```
    DWORD        dwInBufferCount;
```

```
    DWORD        dwInBufferIndex;
```

30

```
    DWORD        dwOutBufferCount;
```

```
    DWORD        dwFrame;
```

```
    DWORD        dwSubFrame;
```

```
    DWORD        dwFrameLength;
```

```
    DWORD        dwDataLength;
```

```

        DWORD      dwBlockNumber;
        DWORD      dwPreviousBlockNumber;
        DWORD      dwBlockNumberC;
        DWORD      dwBlockStart;
5         DWORD      dwNackCount;
    } PluginInstance;

```

```

// Frame type used to control FSM

```

```

10  #define CP_NOFRAME 2000
    #define CP_UPLOAD 2001
    #define CP_DATA 2002
    #define CP_FINAL 2003

```

```

15  // Subframe type used to control FSM

```

```

    #define CPB_NONE 1000
    #define CPB_FRAMESTART 1001
    #define CPB_UIFRAME 1002
20  #define CPB_MSBFRAMELENGTH 1003
    #define CPB_LSBFRAMELENGTH 1004
    #define CPB_STREAMID 1005
    #define CPB_COMMAND 1006
    #define CPB_NOP 1007

```

```

25  #define CPB_MSBDATALENGTH 1008
    #define CPB_SMSBDATALENGTH 1009
    #define CPB_SLSBDATALENGTH 1010
    #define CPB_LSBDATALENGTH 1011
    #define CPB_MSBBLOCKNUMBER 1012
30  #define CPB_LSBBLOCKNUMBER 1013
    #define CPB_MSBBLOCKNUMBERC 1014
    #define CPB_LSBBLOCKNUMBERC 1015
    #define CPB_DATA 1016
    #define CPB_ESCDATA 1017

```

```

#define CPB_EOT 1018
#define CPB_CRC1 1019
#define CPB_CRC2 1020
#define CPB_FRAMEEND 1021
5  #define CPB_FL0 1022

```

```

// Constants defined by IBM's communications protocol

```

```

#define CP_ESCAPE 0x7D
10 #define CP_FRAME_START 0xC0
#define CP_UI_FRAME 0xA
#define CP_GET_SET 0x3
#define CP_STREAM 2
#define CP_FRAME_END 0xC1
15 #define CP_NEGOTIATE_ID 0
#define CP_BYTE_VERB 2
#define CP_RESPONSE 0
#define CP_SUCCESS 0x65
#define CP_ACK 6
20 #define CP_NACK 0x15
#define CP_EOT 7
#define CP_BEGIN_STREAM 3000
#define CP_NOOP 0
#define CP_NOP 2
25
#define uWORD                unsigned int
#define uBYTE                unsigned char
#define initialCrcValue (uWORD)0xFFFF
#define goodCrcValue  (uWORD)0xF0B8
30
/*-----
*/
/* CRC-16 lookup table
*/

```

```
/*-----  
-*/
```

```
const uWORD crcLookupTable[256]=
```

```
{
```

```
5  0x0000,0x1189,0x2312,0x329b,0x4624,0x57ad,0x6536,0x74bf,  
    0x8c48,0x9dc1,0xaf5a,0xbed3,0xca6c,0xdbe5,0xe97e,0xf8f7,  
    0x1081,0x0108,0x3393,0x221a,0x56a5,0x472c,0x75b7,0x643e,  
    0x9cc9,0x8d40,0xbfdb,0xae52,0xdaed,0xcb64,0xf9ff,0xe876,  
    0x2102,0x308b,0x0210,0x1399,0x6726,0x76af,0x4434,0x55bd,  
10 0xad4a,0xbcc3,0x8e58,0x9fd1,0xeb6e,0xfae7,0xc87c,0xd9f5,  
    0x3183,0x200a,0x1291,0x0318,0x77a7,0x662e,0x54b5,0x453c,  
    0xbdc b,0xac42,0x9ed9,0x8f50,0xfbef,0xea66,0xd8fd,0xc974,  
    0x4204,0x538d,0x6116,0x709f,0x0420,0x15a9,0x2732,0x36bb,  
    0xce4c,0xdfc5,0xed5e,0xfcd7,0x8868,0x99e1,0xab7a,0xbaf3,  
15 0x5285,0x430c,0x7197,0x601e,0x14a1,0x0528,0x37b3,0x263a,  
    0xdecd,0xcf44,0xfddf,0xec56,0x98e9,0x8960,0xbbfb,0xaa72,  
    0x6306,0x728f,0x4014,0x519d,0x2522,0x34ab,0x0630,0x17b9,  
    0xef4e,0xfec7,0xcc5c,0xddd5,0xa96a,0xb8e3,0x8a78,0x9bf1,  
    0x7387,0x620e,0x5095,0x411c,0x35a3,0x242a,0x16b1,0x0738,  
20 0xffcf,0xee46,0xdcdd,0xcd54,0xb9eb,0xa862,0x9af9,0x8b70,  
    0x8408,0x9581,0xa71a,0xb693,0xc22c,0xd3a5,0xe13e,0xf0b7,  
    0x0840,0x19c9,0x2b52,0x3adb,0x4e64,0x5fed,0x6d76,0x7cff,  
    0x9489,0x8500,0xb79b,0xa612,0xd2ad,0xc324,0xf1bf,0xe036,  
    0x18c1,0x0948,0x3bd3,0x2a5a,0x5ee5,0x4f6c,0x7df7,0x6c7e,  
25 0xa50a,0xb483,0x8618,0x9791,0xe32e,0xf2a7,0xc03c,0xd1b5,  
    0x2942,0x38cb,0x0a50,0x1bd9,0x6f66,0x7eef,0x4c74,0x5dfd,  
    0xb58b,0xa402,0x9699,0x8710,0xf3af,0xe226,0xd0bd,0xc134,  
    0x39c3,0x284a,0x1ad1,0x0b58,0x7fe7,0x6e6e,0x5cf5,0x4d7c,  
    0xc60c,0xd785,0xe51e,0xf497,0x8028,0x91a1,0xa33a,0xb2b3,  
30 0x4a44,0x5bcd,0x6956,0x78df,0x0c60,0x1de9,0x2f72,0x3efb,  
    0xd68d,0xc704,0xf59f,0xe416,0x90a9,0x8120,0xb3bb,0xa232,  
    0x5ac5,0x4b4c,0x79d7,0x685e,0x1ce1,0x0d68,0x3ff3,0x2e7a,  
    0xe70e,0xf687,0xc41c,0xd595,0xa12a,0xb0a3,0x8238,0x93b1,  
    0x6b46,0x7acf,0x4854,0x59dd,0x2d62,0x3ceb,0x0e70,0x1ff9,
```

```

0xf78f,0xe606,0xd49d,0xc514,0xb1ab,0xa022,0x92b9,0x8330,
0x7bc7,0x6a4e,0x58d5,0x495c,0x3de3,0x2c6a,0x1ef1,0x0f78
};

```

```

5  /*

```

```

    Cleanup - Initialize communications variables for the instance

```

```

    */

```

```

void Cleanup(PluginInstance* This)

```

```

{

```

```

10

```

```

    This->dwInBufferCount=0;

```

```

    This->dwInBufferIndex=0;

```

```

    This->dwOutBufferCount=0;

```

```

    This->dwFrame=CP_NOFRAME;

```

```

    This->dwSubFrame=CPB_NONE;

```

```

15

```

```

    This->dwFrameLength=0;

```

```

    This->dwDataLength=0;

```

```

    This->dwBlockNumber=0;

```

```

    This->dwPreviousBlockNumber=-1;

```

```

    This->dwBlockNumberC=0;

```

```

20

```

```

    This->dwBlockStart=0;

```

```

    This->dwNackCount=0;

```

```

    This->gReading=FALSE;

```

```

}

```

```

25

```

```

    /*-----

```

```

    -*/

```

```

    /* CrcCalculate      Calculate a new CRC given the current

```

```

    */

```

```

    /*                  CRC and the new data.

```

```

30

```

```

    */

```

```

    /*-----

```

```

    -*/

```

```

uWORD CrcCalculate

```

```

    ( uWORD oldCrc,      /* in: CRC calculated "so far" */

```

```

    uBYTE Data)      /* in: data byte to calculate CRC on */
{
    uWORD newCrc = oldCrc;

5    newCrc = (oldCrc >> 8) ^ crcLookupTable[(oldCrc ^ Data) & 0xff];

    return newCrc;
}

```

```

10  /*-----
    *
    * calculateCrc      Calculate a new CRC given the current CRC and * the new data.
    *
15  *-----
    */

```

```

uWORD calculateCrc
( uWORD oldCrc,      /* in: CRC calculated "so far" */
  uBYTE* pData, /* in: data bytes to calculate CRC on */
20  uWORD len) /* in: number of data bytes */
{
    register uWORD newCrc = oldCrc;

    while (len--)
25    newCrc = (newCrc >> 8) ^ crcLookupTable[(newCrc ^ *pData++) & 0xff];

    return newCrc;

} /* calculateCrc */

```

```

30
uWORD checkCrc(uWORD length, uBYTE * buffer)
{
    uWORD CRC = initialCrcValue;
    CRC = calculateCrc(CRC, buffer, length);
}

```

```
if (CRC == goodCrcValue)
```

```
    return 1;
```

```
return 0;
```

```
}
```

5

```
void DoAck(HANDLE hComm) {
```

```
    unsigned char ackBuffer[9];
```

```
    DWORD dwWritten;
```

```
    ackBuffer[0] = CP_FRAME_START;
```

10

```
    ackBuffer[1] = CP_UI_FRAME;
```

```
    ackBuffer[2] = 0; //Length
```

```
    ackBuffer[3] = 1;
```

```
    ackBuffer[4] = CP_STREAM;
```

```
    ackBuffer[5] = CP_ACK;
```

15

```
    ackBuffer[6] = 0x85; //CRC 1
```

```
    ackBuffer[7] = 0x8F; //CRC 2
```

```
    ackBuffer[8] = CP_FRAME_END;
```

```
    WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

20

```
    WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
```

25

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[0],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[1],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

30

```
    WriteFile(hComm,&ackBuffer[2],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[3],1,&dwWritten,NULL);
```

```
    Sleep(10);
```

```
    WriteFile(hComm,&ackBuffer[4],1,&dwWritten,NULL);
```

```

Sleep(10);
WriteFile(hComm,&ackBuffer[5],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&ackBuffer[6],1,&dwWritten,NULL);
5 Sleep(10);
WriteFile(hComm,&ackBuffer[7],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&ackBuffer[8],1,&dwWritten,NULL);
}

```

```

10 void DoNack(HANDLE hComm) {
    unsigned char nackBuffer[9];
    DWORD dwWritten;
    nackBuffer[0] = CP_FRAME_START;
    15 nackBuffer[1] = CP_UI_FRAME;
    nackBuffer[2] = 0; //Length
    nackBuffer[3] = 1;
    nackBuffer[4] = CP_STREAM;
    nackBuffer[5] = CP_NACK;
    20 nackBuffer[6] = 0x9F; //CRC 1
    nackBuffer[7] = 0xAD; //CRC 2
    nackBuffer[8] = CP_FRAME_END;
    WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
    Sleep(10);
    25 WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
    Sleep(10);
    WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
    Sleep(10);
    WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
    30 Sleep(10);
    WriteFile(hComm,&nackBuffer[0],1,&dwWritten,NULL);
    Sleep(10);
    WriteFile(hComm,&nackBuffer[1],1,&dwWritten,NULL);
    Sleep(10);

```

```

WriteFile(hComm,&nackBuffer[2],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&nackBuffer[3],1,&dwWritten,NULL);
Sleep(10);
5 WriteFile(hComm,&nackBuffer[4],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&nackBuffer[5],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&nackBuffer[6],1,&dwWritten,NULL);
10 Sleep(10);
WriteFile(hComm,&nackBuffer[7],1,&dwWritten,NULL);
Sleep(10);
WriteFile(hComm,&nackBuffer[8],1,&dwWritten,NULL);

```

```

}

```

```

15 BYTE TranslateDigitHex(unsigned char b) {

```

```

    switch (b) {

```

```

        case 0 :

```

```

            return '0';

```

```

20        case 1 :

```

```

            return '1';

```

```

        case 2 :

```

```

            return '2';

```

```

        case 3 :

```

```

25        return '3';

```

```

        case 4 :

```

```

            return '4';

```

```

        case 5 :

```

```

            return '5';

```

```

30        case 6 :

```

```

            return '6';

```

```

        case 7 :

```

```

            return '7';

```

```

        case 8 :

```

```

        return '8';
    case 9 :
        return '9';
    case 10 :
5       return 'a';
    case 11 :
        return 'b';
    case 12 :
        return 'c';
10     case 13 :
        return 'd';
    case 14 :
        return 'e';
    case 15 :
15     return 'f';
    default:
        return '0';
    }
}

```

```

20
int instr(char *str1, char *str2) {
    int i1=0;
    int i2=0;
    int l=strlen(str2);
25     do {
        if (str1[i1++]==str2[i2++]) {
            if (i2==l) return 1;
        } else {
            if (str1[i1]=='\0') return 0;
30             i2=0;
        }
    } while (1);
}

```

```
void AddMessage( HWND hwnd, char* message )
```

```
{
```

```
    int i;
```

```
5        if( gMessageTextIndex >= gNumLines )        // If exceeded preset line
number display, reset to first line.
```

```
    {
```

```
        // Clear array and reset counter
```

```
10        for ( i = 0; i < gNumLines; i++ )
```

```
    {
```

```
        strcpy( gMessageTextArray[i], "" );
```

```
    }
```

```
15        gMessageTextIndex = 0;
```

```
    }
```

```
        strcpy( gMessageTextArray[gMessageTextIndex++], message );
```

```
20        if( hwnd )        // So messages can be collected while a valid window
handle hasn't been declared.
```

```
    {
```

```
        InvalidateRect( hwnd, NULL, TRUE );
```

```
        UpdateWindow( hwnd );
```

```
25    }
```

```
}
```

```
NPError NPP_Initialize(void) {
```

```
#ifdef _DEBUG
```

```
30    {
```

```
        char str[100];
```

```
        sprintf(str, "NPP_Initialize\r\n");
```

```
        OutputDebugString(str);
```

```
    }
```

#endif

```
gConnected = 0;
outBufferSize = 100000;
5  inBufferSize = outBufferSize*2+2;
    inBuffer = (unsigned char *) NPN_MemAlloc(inBufferSize);
    outBuffer = (unsigned char *) NPN_MemAlloc(outBufferSize);
    if ((inBuffer==NULL) || (outBuffer==NULL)) {
        if (inBuffer) {
10         NPN_MemFree(inBuffer);
            inBuffer = NULL;
        }
        if (outBuffer) {
            NPN_MemFree(outBuffer);
15         outBuffer = NULL;
        }
        return NPERR_OUT_OF_MEMORY_ERROR;
    }
    return NPERR_NO_ERROR;
20 }
```

```
jref NPP_GetJavaClass(void) {
    return NULL;
}
```

25

```
// Deallocate I/O buffers and close the COM port void NPP_Shutdown(void) {
```

```
    // Close the comm connection;
```

```
30 #ifdef _DEBUG
```

```
{
```

```
    char str[100];
```

```
    sprintf(str, "NPP_Shutdown gConnected=%d
```

```
hComm=%8.8lx\r\n", gConnected, hComm);
```

```

        OutputDebugString(str);
    }
#endif

```

```

5      if (gConnected)
        CloseHandle(hComm);
        gConnected=0;

        // Free memory.
10     if (inBuffer!=NULL) NPN_MemFree(inBuffer);
        inBuffer=NULL;
        if (outBuffer!=NULL) NPN_MemFree(outBuffer);
        outBuffer=NULL;
15  }

```

```

NPPError NPP_New(NPMIMEType pluginType,
    NPP instance,
    uint16 mode,
20   int16 argc,
    char* argn[],
    char* argv[],
    NPSaveData* saved) {

```

```

25   DCB dcb;
    COMMTIMEOUTS ctm;
    BOOL gSuccess;
    int i;
    NPPError result = NPERR_NO_ERROR;
30   PluginInstance* This;

```

```

#ifdef _DEBUG
    {
        char str[100];

```

```

        sprintf(str,"NPP_New instance=%08.8lx
gConn=%0d\r\n",instance,gConnected);
        OutputDebugString(str);
    }

```

5 #endif

```

        if (instance == NULL) {
            return NPERR_INVALID_INSTANCE_ERROR;
        }

```

10 instance->pdata = NPN\_MemAlloc(sizeof(PluginInstance));

```

This = (PluginInstance*) instance->pdata;
if (This == NULL) {
    return NPERR_OUT_OF_MEMORY_ERROR;
}

```

15 /\* mode is NP\_EMBED, NP\_FULL, or NP\_BACKGROUND (see npapi.h) \*/

```

This->fWindow = NULL;
This->fMode = mode;
This->fhWnd = NULL;
This->fDefaultWindowProc = NULL;

```

20

```

// Initialize communications variables
Cleanup(This);

```

// Save plug-in instance

25

```

This->gInstance = instance;

```

```

// Get plugin parameters (hostname,hostport,uid,proxynome,proxyport,
// comm port, baud rate, sourceurl) that
30 // was passed into the plugin via html.

```

```

This->gHostName[0] = '\0';
This->gHostPort[0] = '\0';
This->gUID[0] = '\0';

```

This->gProxyName[0] = '\0';

This->gProxyPort[0] = '\0';

This->gComPort[0] = '\0';

This->gComSpeed[0] = '\0';

5 This->gSourceURL[0] = '\0';

This->gVerbose = FALSE;

This->gVersion[0] = '\0';

for (i=0; i<argc; i++) {

10 if (strcmp(strupr(argn[i]),"HOSTNAME")==0) {  
strcpy( This->gHostName, argv[i]);

} else if (strcmp(strupr(argn[i]),"HOSTPORT")==0) {  
strcpy( This->gHostPort, argv[i]);

15 } else if (strcmp(strupr(argn[i]),"UID")==0) {  
strcpy( This->gUID, argv[i]);

} else if (strcmp(strupr(argn[i]),"PROXYNAME")==0) {  
strcpy( This->gProxyName, argv[i]);

} else if (strcmp(strupr(argn[i]),"PROXYPORT")==0) {  
20 strcpy( This->gProxyPort, argv[i]);

} else if (strcmp(strupr(argn[i]),"COMPORT")==0) {  
strcpy( This->gComPort, argv[i]);

} else if (strcmp(strupr(argn[i]),"COMSPEED")==0) {  
strcpy( This->gComSpeed, argv[i]);

25 } else if (strcmp(strupr(argn[i]),"SOURCEURL")==0) {  
strcpy( This->gSourceURL, argv[i]);

} else if (strcmp(strupr(argn[i]),"NUMLINES")==0) {  
gNumLines=atoi( argv[i] );

30 } else if (strcmp(strupr(argn[i]),"VERBOSE")==0) {  
This->gVerbose=TRUE;

} else if (strcmp(strupr(argn[i]),"VERSION")==0) {  
strcpy( This->gVersion, argv[i]);

}

}

```
// Close the comm connection so that the port parameters can be  
reset
```

```
    if (gConnected)    {  
5        CloseHandle(hComm);  
    }
```

```
#ifdef _DEBUG
```

```
{  
10    char str[100];  
    sprintf(str,"Closed comm port instance=%08lx  
gConn=%d\r\n",instance,gConnected);  
    OutputDebugString(str);  
}
```

```
15 #endif
```

```
// Connect to the Comm port and allocate the buffers.
```

```
// hComm=CreateFile("D:\\TEMP\\Copy (2) of COMMLOG.BIN",GENERIC_READ |  
GENERIC_WRITE ,FILE_SHARE_WRITE,NULL,OPEN_EXISTING,0,NULL);
```

```
20 hComm=CreateFile(This->gComPort,GENERIC_READ | GENERIC_WRITE  
,FILE_SHARE_WRITE,NULL,OPEN_EXISTING,0,NULL);
```

```
    if (hComm==INVALID_HANDLE_VALUE) {  
        char message[256];  
        strcpy( message, "Error connecting to ");  
25        strcat( message, This->gComPort );  
        strcat( message, " - please confirm that it is available" );  
        AddMessage( This->fhWnd, message ); // *****  
        return 0;  
    }
```

```
30    ++gConnected;
```

```
#ifdef _DEBUG
```

```
{  
    char str[100];
```

```

        sprintf(str,"Opening hComm=%8.8lx
gConn=%d\r\n",hComm,gConnected);
        OutputDebugString(str);
    }

```

5 #endif

```

        gSuccess=GetCommState(hComm,&dcb);
        if (!gSuccess) {
            AddMessage( This->fhWnd, "Error on GetCommState()..." );

```

10 // \*\*\*\*\*

```

            return 0;

```

```

        }

```

```

        dcb.DCBlength=sizeof(dcb);

```

```

        dcb.BaudRate=atol( This->gComSpeed );

```

15

```

        dcb.ByteSize=8;

```

```

        dcb.Parity=NOPARITY;

```

```

        dcb.StopBits=ONESTOPBIT;

```

```

        dcb.fBinary=1;

```

```

        gSuccess=SetCommState(hComm,&dcb);

```

20

```

        if (!gSuccess) {

```

```

            AddMessage( This->fhWnd, "Error on GetCommState()..." );

```

// \*\*\*\*\*

```

            return 0;

```

```

        }

```

25

```

        ctm.ReadIntervalTimeout=MAXDWORD;

```

```

        ctm.ReadTotalTimeoutConstant=0;

```

```

        ctm.ReadTotalTimeoutMultiplier=0;

```

```

        ctm.WriteTotalTimeoutConstant=0;

```

```

        ctm.WriteTotalTimeoutMultiplier=0;

```

30

```

        gSuccess=SetCommTimeouts(hComm,&ctm);

```

```

        if (!gSuccess) {

```

```

            AddMessage( This->fhWnd, "Error on SetCommTimeouts()..." );

```

// \*\*\*\*\*

```

            return 0;

```

```

    }
    {
        char message[256];
        sprintf(message,"Connected to %s - please initiate upload from
5  pad...",This->gComPort);
        AddMessage( This->fhWnd, message ); // *****
    }
    Cleanup(This);

10    // Check the version

        if (strcmp(This->gVersion,"1.2.6")!=0) {
            AddMessage( This->fhWnd, "Warning - incorrect version of
plug-in is installed. Please upgrade plug-in..." ); // *****
15    }

#ifdef _DEBUG
    {
        char str[100];
20        sprintf(str,"End of NPP_New instance=%8.8lx
gConn=%d\r\n",instance,gConnected);
        OutputDebugString(str);
    }
#endif

25    return result;
}

NPPError NPP_Destroy(NPP instance, NPSaveData** save) {
30    PluginInstance* This;

#ifdef _DEBUG
    {
        char str[100];

```

```

        sprintf(str,"NPP_Destroy instance=%8.8lx
gCon=%d\r\n",instance,gConnected);
        OutputDebugString(str);
    }

```

5 #endif

```

    if (instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

```

```

10     This = (PluginInstance*) instance->pdata;
    if (This != NULL) {
        // Kill the timer.

```

```

#ifdef _DEBUG

```

```

15     {
        char str[100];
        sprintf(str,"Destroy timer %8.8lx\r\n",This->fhWnd);
        OutputDebugString(str);
    }

```

20 #endif

```

    KillTimer(This->fhWnd, 1);

```

```

    if( This->fWindow != NULL ) {
25        SetWindowLong( This->fhWnd, GWL_WNDPROC,
        (LONG)This->fDefaultWindowProc);
        This->fDefaultWindowProc = NULL;
        This->fhWnd = NULL;
    }

```

30

```

    NPN_MemFree(instance->pdata);
    instance->pdata = NULL;
}

```

```
// Close the comm connection on the last instance only
```

```
if (gConnected == 1)
{
    CloseHandle(hComm);
```

5

```
}
--gConnected;
```

```
return NPERR_NO_ERROR;
```

10 }

```
NPError NPP_SetWindow(NPP instance, NPWindow* window) {
```

```
    NPError result = NPERR_NO_ERROR;
```

```
    PluginInstance* This;
```

15

```
    HWND hButton;
```

```
    HANDLE hImage;
```

```
    HANDLE hInstance;
```

```
    RECT rect;
```

20

```
    if (instance == NULL)
```

```
        return NPERR_INVALID_INSTANCE_ERROR;
```

```
    This = (PluginInstance*) instance->pdata;
```

25 #ifdef \_DEBUG

```
{
```

```
    char str[100];
```

```
    sprintf(str, "NPP_SetWindow
```

```
instance=%8.8lx(%8.8lx)\r\n", instance, This->gInstance);
```

30

```
    OutputDebugString(str);
```

```
}
```

```
#endif
```

```
if( This->fWindow != NULL ) /* If we already have a window, clean
```

\* it up

before trying to subclass

\* the new

window. \*/

```
5      {  
        if( (window == NULL) || ( window->window == NULL ) ) {  
          /* There is now no window to use. get rid of the old  
           * one and exit. */  
          SetWindowLong( This->fhWnd, GWL_WNDPROC,  
10      (LONG)This->fDefaultWindowProc);  
          This->fDefaultWindowProc = NULL;  
          This->fhWnd = NULL;  
          This->fWindow=window;  
          return NPERR_NO_ERROR;  
15      }  
  
          else if ( This->fhWnd == (HWND) window->window ) {  
            /* The new window is the same as the old one. Redraw  
and get out. */  
20      InvalidateRect( This->fhWnd, NULL, TRUE );  
            UpdateWindow( This->fhWnd );  
            This->fWindow=window;  
            return NPERR_NO_ERROR;  
          }  
25      else {  
            /* Clean up the old window, so that we can subclass  
the new  
            * one later. */  
            SetWindowLong( This->fhWnd, GWL_WNDPROC,  
30      (LONG)This->fDefaultWindowProc);  
            This->fDefaultWindowProc = NULL;  
            This->fhWnd = NULL;  
          }  
      }  
    }
```

```
else if( (window == NULL) || ( window->window == NULL ) ) {
```

```
    /* We can just get out of here if there is no current  
    * window and there is no new window to use. */
```

```
    This->fWindow=window;
```

```
5      return NPERR_NO_ERROR;
```

```
    }
```

```
    /* At this point, we will subclass
```

```
    * window->window so that we can begin drawing and
```

```
10    * receiving window messages. */
```

```
#ifdef _DEBUG
```

```
{
```

```
    char str[200];
```

```
    sprintf(str,"Subclassing window %8.8lx fhWnd =
```

```
15    %8.8lx\r\n",window->window,This->fhWnd);
```

```
    OutputDebugString(str);
```

```
}
```

```
#endif
```

```
20    This->fDefaultWindowProc = (WNDPROC)SetWindowLong(  
(HWND)window->window, GWL_WNDPROC, (LONG)PluginWindowProc);
```

```
    This->fhWnd = (HWND) window->window;
```

```
    SetProp( This->fhWnd, gInstanceLookupString, (HANDLE)This);
```

```
25    try
```

```
{
```

```
        IApplicationPtr pApp(__uuidof(Application));
```

```
        This->bTransNote = TRUE;
```

```
30    }
```

```
    catch(...)
```

```
{
```

```
        This->bTransNote = FALSE;
```

```
}
```

// Create button

GetClientRect(This->hWnd,&rect);

gNumLines = rect.bottom/20;

if (This->bTransNote)

5 { hInstance = (HANDLE)

GetWindowLong(This->hWnd,GWL\_HINSTANCE);

hButton = CreateWindow("button","IBM Upload",WS\_CHILD |  
WS\_BORDER | WS\_VISIBLE | BS\_PUSHBUTTON | BS\_CENTER | BS\_BITMAP |  
BS\_VCENTER,

10 rect.right-120,0,120,32,  
This->hWnd,(HMENU) 1,(HINSTANCE)

hInstance,NULL);

hImage =

LoadImage(GetModuleHandle("NPTimbrl.dll"),MAKEINTRESOURCE(IDB\_WORKONCE),IMAG  
15 E\_BITMAP,0,0,LR\_SHARED);

if (hImage)

SendMessage(hButton,BM\_SETIMAGE,IMAGE\_BITMAP,(LONG)

hImage);

}

20 #ifdef \_DEBUG

else

{ hInstance = (HANDLE)

GetWindowLong(This->hWnd,GWL\_HINSTANCE);

hButton = CreateWindow("button","Upload File",WS\_CHILD |  
25 WS\_BORDER | WS\_VISIBLE | BS\_PUSHBUTTON | BS\_CENTER | BS\_VCENTER,  
rect.right-90,0,90,30,  
This->hWnd,(HMENU) 1,(HINSTANCE)

hInstance,NULL);

}

30 #endif

// Create timer for window

#ifdef \_DEBUG

{

```

char str[100];
sprintf(str,"Create timer %8.8lx\r\n",This->fhWnd);
OutputDebugString(str);
}

```

5 #endif

```

SetTimer( This->fhWnd, 1, 0, NULL );

```

```

InvalidateRect( This->fhWnd, NULL, TRUE );

```

10 UpdateWindow( This->fhWnd );

```

This->fWindow = window;

```

```

return result;
}

```

15

```

NPPError NPP_NewStream(NPP instance,

```

```

    NPMIMEType type,

```

```

    NPStream *stream,

```

```

    NPBool seekable,

```

20 uint16 \*stype) {

```

    PluginInstance* This;

```

```

    if (instance == NULL)

```

```

        return NPERR_INVALID_INSTANCE_ERROR;

```

25

```

    This = (PluginInstance*) instance->pdata;

```

```

    return NPERR_NO_ERROR;
}

```

30

```

int32 STREAMBUFSIZE = 0X0FFFFFFF; /* If we are reading from a file in
NPAsFile

```

\* mode so

we can take any size stream in our

\* write

call (since we ignore it) \*/

```
int32 NPP_WriteReady(NPP instance, NPStream *stream) {
```

```
5     PluginInstance* This;
```

```
    if (instance != NULL)
```

```
        This = (PluginInstance*) instance->pdata;
```

```
    return STREAMBUFSIZE;
```

```
}
```

```
10
```

```
int32 NPP_Write(NPP instance, NPStream *stream, int32 offset, int32 len,  
void *buffer) {
```

```
    if (instance != NULL) {
```

```
        PluginInstance* This = (PluginInstance*) instance->pdata;
```

```
15    }
```

```
    return len;          /* The number of bytes accepted */
```

```
}
```

```
NPError NPP_DestroyStream(NPP instance, NPStream *stream, NPError reason) {
```

```
20
```

```
    PluginInstance* This;
```

```
    if (instance == NULL)
```

```
        return NPERR_INVALID_INSTANCE_ERROR;
```

```
    This = (PluginInstance*) instance->pdata;
```

```
25
```

```
    return NPERR_NO_ERROR;
```

```
}
```

```
void NPP_StreamAsFile(NPP instance, NPStream *stream, const char* fname) {
```

```
30
```

```
    PluginInstance* This;
```

```
    if (instance != NULL)
```

```
        This = (PluginInstance*) instance->pdata;
```

```
}
```

```

void NPP_Print(NPP instance, NPPrint* printInfo) {
    if(printInfo == NULL)
        return;

```

```

5      if (instance != NULL) {
        PluginInstance* This = (PluginInstance*) instance->pdata;

        if (printInfo->mode == NP_FULL) {

```

```

10         void* platformPrint =
            printInfo->print.fullPrint.platformPrint;
            NPBool printOne =
                printInfo->print.fullPrint.printOne;

15         /* Do the default*/
            printInfo->print.fullPrint.pluginPrinted = FALSE;
        }
        else { /* If not fullscreen, we must be embedded */
            NPWindow* printWindow =
20                &(printInfo->print.embedPrint.window);
            void* platformPrint =
                printInfo->print.embedPrint.platformPrint;
        }
    }
25 }

```

```

void NPP_URLNotify( NPP instance, const char* url, NPReason reason, void*
notifyData ) {
    switch( reason ) {
30      case NPRES_DONE:          // Completed normally.
        break;
        case NPRES_USER_BREAK: // User canceled stream directly or
indirectly.
        break;

```

```
        case NPRES_NETWORK_ERR: // Stream failed due to problems with
network, disk I/O, lack of memory, or other problems.
```

```
            break;
```

```
        }
```

```
5    }
```

```
int16 NPP_HandleEvent(NPP instance, void* event)
```

```
{
```

```
    return 0;
```

```
10 }
```

```
int PostURL(HWND hWnd, char *hostname, unsigned short hostport, int uid,
unsigned char* buffer, int bufferlen, char *proxyname, unsigned short
proxypart) {
```

```
15     SOCKET skt;
```

```
     INT iResult;
```

```
     SOCKADDR_IN server;
```

```
     WSADATA wsaData;
```

```
     HOSTENT *host;
```

```
20     BOOL useproxy=0;
```

```
     int cc=0;
```

```
     char httpBuffer[256];
```

```
     PluginInstance* This = (PluginInstance*) GetProp(hWnd,
glInstanceLookupString);
```

```
25
```

```
     iResult = WSASStartup(0x202,&wsaData);
```

```
     if (iResult==SOCKET_ERROR) {
```

```
         sprintf(httpBuffer,"Error on %d
```

```
WSASStartup()...",WSAGetLastError());
```

```
30
```

```
         AddMessage(hWnd,httpBuffer);
```

```
         return -1;
```

```
     }
```

```
     skt=socket(AF_INET,SOCK_STREAM,0);
```

```
     if (skt<0) {
```

```

        sprintf(httpBuffer,"Error %d on
socket(...",WSAGetLastError());
        AddMessage(hWnd,httpBuffer);
        return -1;

```

```

5      }
      if (strcmp(proxyname,"")!=0) useproxy=1;
      if (useproxy==1) {

```

```

        sprintf(httpBuffer,"Looking up proxy %s...",proxyname);
        if (This->gVerbose) AddMessage(hWnd,httpBuffer);
10     host=gethostbyname(proxyname);
        server.sin_port=htons(proxyport);

```

```

      } else {
        sprintf(httpBuffer,"Looking up host %s...",hostname);
        if (This->gVerbose) AddMessage(hWnd,httpBuffer);
15     host=gethostbyname(hostname);
        server.sin_port=htons(hostport);
      }

```

```

      if (host==NULL) {
        sprintf(httpBuffer,"Error %d on
20    gethostbyname(...",WSAGetLastError());
        AddMessage(hWnd,httpBuffer);
        return -1;
      }

```

```

      memcpy(&(server.sin_addr),*host->h_addr_list,host->h_length);
25     server.sin_family=host->h_addrtype;
      if (This->gVerbose) AddMessage(hWnd,"Connecting...");
      //iResult=connect(skt,(SOCKADDR*)&server,sizeof(server));
      //if (iResult==SOCKET_ERROR) {
      //    sprintf(httpBuffer,"Error %d on
30    connect(...",WSAGetLastError());
      //    AddMessage(hWnd,httpBuffer);
      //    return -1;
      //}
      // Try connecting multiple times - this exists to help us manage

```

```

// peak OCRServer traffic while we scale. If no processes
// are available to service this connection, try again a number of
// times.
for (cc=0;cc<10;cc++) {
5         iResult=connect(skt,(SOCKADDR*)&server,sizeof(server));
        if (iResult!=SOCKET_ERROR) break;
        Sleep(100);
    }
    if (cc==10) {
10        sprintf(httpBuffer,"Server Busy - Please Try Again...");
        AddMessage(hWnd,httpBuffer);
        return -1;
    }
    if (This->gVerbose) AddMessage(hWnd,"Executing HTTP POST
15 method...");
    if (useproxy==1) {
        if (hostport!=80) {
            sprintf(httpBuffer,"POST http://%s:%d/%d/
HTTP/1.0\nContent-Type: application/x-www-form-urlencoded\nContent-Length:
20 %d\n\n",hostname,hostport,uid,bufferlen);
        } else {
            sprintf(httpBuffer,"POST http://%s/%d/
HTTP/1.0\nContent-Type: application/x-www-form-urlencoded\nContent-Length:
%d\n\n",hostname,uid,bufferlen);
25        }
    } else {
        sprintf(httpBuffer,"POST %d \nContent-Type:
application/x-www-form-urlencoded\nContent-Length: %d\n\n",uid,bufferlen);
    }
30    iResult=send(skt,(const char*)httpBuffer,strlen(httpBuffer),0);
    iResult=send(skt,(const char*)buffer,bufferlen,0);
    if (This->gVerbose) AddMessage(hWnd,"Waiting on HTTP response...");
    iResult=recv(skt,httpBuffer,sizeof(httpBuffer),0);
    closesocket(skt);

```

WSACleanup();

if (This->gVerbose) AddMessage(hWnd,"Socket closed...");

if (iResult==SOCKET\_ERROR) {

5           AddMessage(hWnd,"Error on recv(...)");

          return -1;

  } else if (iResult==0) {

          AddMessage(hWnd,"Error on recv(...)");

          return -1;

10       } else {

          httpBuffer[iResult]='\0';

          if (This->gVerbose) AddMessage(hWnd,"Received HTTP  
response...");

      }

15       if (instr(httpBuffer,"<body>OK</body>")==1) {

          return 0;

      } else {

          if (This->gVerbose) AddMessage(hWnd,httpBuffer);

          return -1;

20       }

  }

char\* AddTick(char \*str) {

  static int tickCount;

25    int i;

  if (str==NULL) {

    tickCount=0;

    return NULL;

  } else {

30      tickCount++;

    sprintf(str,"Uploading");

    for (i=0;i<tickCount;i++)

      str[9+i]='.';

    str[9+i]='\0';

```

        return str;
    }
}

```

5 uBYTE GetNextByte(unsigned char \*inBuffer,DWORD \*index)

```

{
    uBYTE result;

    result = inBuffer[*index];
10    ++*index;
    if (result == CP_ESCAPE) {
        result = inBuffer[*index] ^ 0x20;
        ++*index;
    }
15    return result;
}

```

LRESULT CALLBACK PluginWindowProc( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)

```

20 {
    PluginInstance* This = (PluginInstance*) GetProp(hWnd,
gInstanceLookupString);

    PAINTSTRUCT paintStruct;
25    HDC          hdc;

    //static unsigned char inBuffer[150000];
    //static unsigned char outBuffer[75000];
    DWORD dwRead;
30    BOOL fDone=FALSE;
    char message[256];
    static uWORD CRC=0;
    unsigned char b2=0;
    DWORD i2=0;

```

```

        DWORD i1=0;
        NPErrors err;
        int i;
        BOOL gSuccess;
5      HWND hButton;
        RECT rect;
        long w,h;
        HANDLE hFile;
        _bstr_t szFileName;
10
        switch( Msg ) {
            case WM_SIZE:
                hButton = GetDlgItem(hWnd,1);
                if (IsWindow(hButton))
15          {
                    GetWindowRect(hButton,&rect);
                    w = rect.right-rect.left;
                    h = rect.bottom-rect.top;
                    GetClientRect(hWnd,&rect);

#ifdef _DEBUG
20          {
                        char str[100];
                        sprintf(str,"Rect (%d,%d)-(%d,%d)
Button %d x %d\r\n",
25      rect.left,rect.top,rect.right,rect.bottom,w,h);
                        OutputDebugString(str);
                    }
            #endif

                MoveWindow(hButton,rect.right-w,0,w,h,TRUE);
30          }
                GetClientRect(hWnd,&rect);
                gNumLines = rect.bottom/20;
                break;
            case WM_COMMAND:

```

```

if (This->bTransNote)
{
    try
    {
        IApplicationPtr
5    pApp(__uuidof(Application));
        IArchivePtr    pArchive =
        pApp->Archive;
        //            long lPadNo = pArchive->Count;
        //            IPadInfoPtr    pPadInfo =
10    pArchive->Item[(long) (lPadNo-1)];
        IPadInfoPtr    pPadInfo =
        pArchive->GetActivePad();
        szFileName = pPadInfo->FileName;
    }
15    catch(...)
    {
        AddMessage( This->fhWnd,"Error
accessing COM object");
        break;
20    }
}

#ifdef _DEBUG
    else
    {
        OPENFILENAME ofn;
25
        memset(&ofn,0,sizeof(ofn));
        ofn.lStructSize = sizeof(ofn);
        ofn.hwndOwner = hWnd;
        ofn.lpstrFilter = "Ink Files
30    (*.ixu,*.pad)\0*.ixu;*.pad\0";
        ofn.nFilterIndex = 1;
        message[0] = 0;
        ofn.lpstrFile = message;
        ofn.nMaxFile = 256;

```

```
ofn.Flags = OFN__ENABLESIZING | OFN_EXPLORER  
| OFN_FILEMUSTEXIST;
```

```
if (GetOpenFileName(&ofn))  
    szFileName = ofn.lpstrFile;
```

```
5          else  
            break;
```

```
    }
```

```
#endif
```

```
    hFile =
```

```
10    CreateFile(szFileName,GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,0,NULL  
);
```

```
    if (INVALID_HANDLE_VALUE != hFile)
```

```
    {        This->dwOutBufferCount =
```

```
GetFileSize(hFile,NULL);
```

```
15          if (This->dwOutBufferCount > outBufferSize)
```

```
{
```

```
            unsigned char *tmp;
```

```
            if (This->gVerbose) {
```

```
20                AddMessage(
```

```
This->fhWnd,"Increasing buffer size");
```

```
            }
```

```
            tmp = (unsigned char *)
```

```
25    NPN_MemAlloc(This->dwDataLength);
```

```
            if (tmp == NULL) {
```

```
                AddMessage(This->fhWnd,
```

```
"Unable to reallocate output buffer.");
```

```
                Cleanup(This);
```

```
30            }
```

```
            else {
```

```
memcpy(tmp,outBuffer,outBufferSize);
```

```
            outBufferSize =
```

This->dwDataLength;

NPN\_MemFree(outBuffer);

outBuffer = tmp;

}

5

tmp = (unsigned char \*)

NPN\_MemAlloc(This->dwDataLength\*2+2);

if (tmp == NULL) {

AddMessage(This->fhWnd,

10 "Unable to reallocate input buffer.");

Cleanup(This);

}

else {

15 memcpy(tmp,inBuffer,inBufferSize);

inBufferSize =

This->dwDataLength\*2+2;

NPN\_MemFree(inBuffer);

inBuffer = tmp;

20

}

}

ReadFile(hFile,outBuffer,This->dwOutBufferCount,(unsigned long \*) &w,NULL);

CloseHandle(hFile);

25

sprintf(message,"Read %d bytes from

%",This->dwOutBufferCount,This->gComPort);

if (This->gVerbose) AddMessage( This->fhWnd,

message ); // \*\*\*\*\*

strcpy((char \*) inBuffer,"d=");

30

i2=2;

for (; i1<This->dwOutBufferCount;

i1++,i2+=2) {

b2=(unsigned

char)(outBuffer[i1]>>4);

inBuffer[i2]=TranslateDigitHex(b2);

inBuffer[i2+1]=TranslateDigitHex((unsigned char)(outBuffer[i1]-(b2<<4)));  
}

5

err = PostURL(hWnd,This->gHostName,(unsigned  
short)atoi(This->gHostPort),atoi(This->gUID),inBuffer,This->dwOutBufferCount  
\*2+2,This->gProxyName,(unsigned short)atoi(This->gProxyPort));

if (err==0) {

10

AddMessage( This->fhWnd,"Upload  
Successful - please wait...");

NPN\_GetURL(This->gInstance,This->gSourceURL,"\_current");

} else {

15

AddMessage( This->fhWnd,"Upload  
Failed");

Cleanup(This);  
fDone=FALSE;  
CRC=0;

20

}

}

break;

case WM\_TIMER:

do {

25

gSuccess=ReadFile(hComm,&inBuffer[This->dwInBufferCount],256,&dwRead,NULL);

if (!gSuccess) {

i = GetLastError();

}

30

if (dwRead>0) {

This->dwInBufferCount+=dwRead;

}

if

(This->dwInBufferIndex<This->dwInBufferCount) {

```

// If escape char is at end of
buffer, wait for more data

        if ((inBuffer[This->dwInBufferIndex]
== CP_ESCAPE) && (This->dwInBufferIndex == This->dwInBufferCount-1))
5           continue;

        switch (This->dwFrame) {
        case CP_NOFRAME :
            switch (This->dwSubFrame) {
            case CPB_NONE :
10                 if
(inBuffer[This->dwInBufferIndex]==CP_FRAME_START) {

                This->dwInBufferIndex++;

15         This->dwSubFrame=CPB_UIFRAME;

                                } else {

                This->dwInBufferIndex++;

                                }

20                 break;
            case CPB_UIFRAME :
                if
(inBuffer[This->dwInBufferIndex]==CP_UI_FRAME) {

                                CRC =

25         initialCrcValue;

                                CRC =

                CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

                This->dwSubFrame=CPB_MSBFRAMELENGTH;

30                 } else {

                                AddMessage(

                This->fhWnd,"ERROR ONE..");

                Cleanup(This);

```

```

        return 0;
    }
    break;
    case CPB_MSBFRAMELENGTH :
5       CRC =
        CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
        This->dwFrameLength
        = b2;

10    This->dwSubFrame=CPB_LSBFRAMELENGTH;

        break;
        case CPB_LSBFRAMELENGTH :
            CRC =
                CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));
15            This->dwFrameLength
            = (This->dwFrameLength << 8) + b2;

            This->dwSubFrame=CPB_STREAMID;

        break;
20        case CPB_STREAMID :
            if
                (inBuffer[This->dwInBufferIndex]==CP_STREAM) {
                    CRC =
                        CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));
25            This->dwSubFrame=CPB_COMMAND;

                } else {
                    AddMessage(
                        This->fhWnd,"Pad error - please erase pad and retry..");
30            Cleanup(This);

                return 0;
            }
            break;

```

case CPB\_COMMAND :

if

(inBuffer[This->dwInBufferIndex]==CP\_NOP) {

5 This->dwFrame=CP\_UPLOAD;

This->dwSubFrame=CPB\_NOP;

} else if

(inBuffer[This->dwInBufferIndex]==CP\_EOT) {

10

This->dwFrame=CP\_FINAL;

This->dwSubFrame=CPB\_EOT;

} else {

15

This->dwFrame=CP\_DATA;

This->dwSubFrame=CPB\_MSBBLOCKNUMBER;

}

20

break;

}

break;

case CP\_UPLOAD :

switch (This->dwSubFrame) {

25

case CPB\_NOP :

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB\_MSBDATALENGTH;

30

break;

case CPB\_MSBDATALENGTH :

CRC =

CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwDataLength =

b2;

This->dwSubFrame=CPB\_SMSBDATALENGTH;

break;

5 case CPB\_SMSBDATALENGTH :

CRC =

CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwDataLength =

(This->dwDataLength << 8) + b2;

10

This->dwSubFrame=CPB\_SLSBDATALENGTH;

break;

case CPB\_SLSBDATALENGTH :

CRC =

15 CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwDataLength =

(This->dwDataLength << 8) + b2;

This->dwSubFrame=CPB\_LSBDATALENGTH;

20

break;

case CPB\_LSBDATALENGTH :

CRC =

CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwDataLength =

25 (This->dwDataLength << 8) + b2;

This->dwSubFrame=CPB\_CRC1;

break;

case CPB\_CRC1 :

30

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB\_CRC2;

break;

case CPB\_CRC2 :

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

5 This->dwSubFrame=CPB\_FRAMEEND;

break;

case CPB\_FRAMEEND :

gSuccess = TRUE;

if

10 (inBuffer[This->dwInBufferIndex]!=CP\_FRAME\_END) {

if

(This->gVerbose) {

AddMessage( This->fhWnd,"Frame End Offset...");

15

}

gSuccess =

FALSE;

} else if

(CRC!=goodCrcValue) {

20

AddMessage(

This->fhWnd,"ERROR FOUR...");

gSuccess =

FALSE;

}

25

if (!gSuccess) {

This->dwSubFrame=CPB\_NONE;

This->dwFrame=CP\_NOFRAME;

30

DoNack(hComm);

if

(This->gVerbose) AddMessage( This->fhWnd,"NACK...");

This->dwNackCount++;

(This->dwNackCount==3) fDone=TRUE;

5

This->dwInBufferIndex++;

This->dwSubFrame=CPB\_NONE;

10

This->dwFrame=CP\_NOFRAME;

{

15

This->fhWnd,"ACK...");

This->fhWnd,AddTick(message));

20

25 (This->dwDataLength > outBufferSize) {

char \*tmp;

30 (This->gVerbose) {

AddMessage( This->fhWnd,"Increasing buffer size");

if

break;

}

DoAck(hComm);

if (This->gVerbose)

AddMessage(

} else {

AddMessage(

}

This->dwNackCount=0;

This->gReading=TRUE;

if

unsigned

if

}

```

tmp =
(unsigned char *) NPN_MemAlloc(This->dwDataLength);
if (tmp ==
NULL) {
5
AddMessage(This->fhWnd, "Unable to reallocate output buffer.");

Cleanup(This);
}
10
else {

memcpy(tmp,outBuffer,outBufferSize);

outBufferSize = This->dwDataLength;
15
NPN_MemFree(outBuffer);

outBuffer = tmp;
}
20
tmp =
(unsigned char *) NPN_MemAlloc(This->dwDataLength*2+2);
if (tmp ==
NULL) {
25
AddMessage(This->fhWnd, "Unable to reallocate input buffer.");

Cleanup(This);
}
30
else {

memcpy(tmp,inBuffer,inBufferSize);

inBufferSize = This->dwDataLength*2+2;

```

NPN\_MemFree(inBuffer);

inBuffer = tmp;

5 }

}

break;

}

10 break;

case CP\_DATA :

switch (This->dwSubFrame) {

case CPB\_MSBBLOCKNUMBER :

CRC =

15 CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwBlockNumber

= b2;

This->dwFrameLength--;

20

This->dwDataLength--;

This->dwSubFrame=CPB\_LSBBLOCKNUMBER;

break;

25 case CPB\_LSBBLOCKNUMBER :

CRC =

CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwBlockNumber

= (This->dwBlockNumber << 8) + b2;

30

This->dwFrameLength--;

This->dwDataLength--;

This->dwSubFrame=CPB\_MSBBLOCKNUMBERC;

break;

case CPB\_MSBBLOCKNUMBERC :

CRC =

5 CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwBlockNumberC

= b2;

This->dwFrameLength--;

10

This->dwDataLength--;

This->dwSubFrame=CPB\_LSBBLOCKNUMBERC;

break;

15

case CPB\_LSBBLOCKNUMBERC :

CRC =

CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwBlockNumberC

= (This->dwBlockNumberC << 8) + b2;

20

This->dwFrameLength--;

This->dwDataLength--;

25 This->dwBlockStart=This->dwOutBufferCount;

This->dwSubFrame=CPB\_DATA;

break;

case CPB\_DATA :

30

if

(inBuffer[This->dwInBufferIndex]==CP\_FRAME\_END) {

This->dwOutBufferCount-=2;

This->dwSubFrame=CPB\_FRAMEEND;

break;

}

CRC =

5 CrcCalculate(CRC, b2 = GetNextByte(inBuffer,&This->dwInBufferIndex));

outBuffer[This->dwOutBufferCount++]=b2;

This->dwFrameLength--;

10

This->dwDataLength--;

break;

case CPB\_FRAMEEND :

gSuccess = TRUE;

15

if

(inBuffer[This->dwInBufferIndex]!=CP\_FRAME\_END) {

AddMessage(

This->fhWnd,"ERROR FIVE");

gSuccess =

20

FALSE;

}

if

(This->dwBlockNumber==This->dwPreviousBlockNumber) {

AddMessage(

25

This->fhWnd,"Block Reread...");

}

if

(CRC!=goodCrcValue || !gSuccess) {

30

This->dwPreviousBlockNumber=This->dwBlockNumber;

sprintf(message,"Block Number:%d

CRC:%d=%d",This->dwBlockNumber,CRC,goodCrcValue);

if

(This->gVerbose) AddMessage( This->fhWnd,message);

This->dwOutBufferCount=This->dwBlockStart;

5 This->dwInBufferIndex++;

This->dwSubFrame=CPB\_NONE;

This->dwFrame=CP\_NOFRAME;

10

DoNack(hComm);

if

(This->gVerbose) AddMessage( This->fhWnd,"NACK...");

15 This->dwNackCount++;

if

(This->dwNackCount==3) fDone=TRUE;

} else {

20 This->dwPreviousBlockNumber=This->dwBlockNumber;

sprintf(message,"Block Number:%d

CRC:%d=%d",This->dwBlockNumber,CRC,goodCrcValue);

if

25 (This->gVerbose) AddMessage( This->fhWnd,message);

This->dwInBufferIndex++;

This->dwSubFrame=CPB\_NONE;

30

This->dwFrame=CP\_NOFRAME;

DoAck(hComm);

if

(This->gVerbose) {

AddMessage( This->fhWnd,"ACK...");

} else {

5

AddMessage( This->fhWnd,AddTick(message));

}

}

break;

10

}

break;

case CP\_FINAL :

switch (This->dwSubFrame) {

15

case CPB\_EOT :

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

This->dwSubFrame=CPB\_CRC1;

20

break;

case CPB\_CRC1 :

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

25 This->dwSubFrame=CPB\_CRC2;

break;

case CPB\_CRC2 :

CRC =

CrcCalculate(CRC, GetNextByte(inBuffer,&This->dwInBufferIndex));

30

This->dwSubFrame=CPB\_FRAMEEND;

break;

case CPB\_FRAMEEND :

gSuccess = TRUE;

```

if
(inBuffer[This->dwInBufferIndex]!=CP_FRAME_END) {
    AddMessage(
    This->fhWnd,"ERROR SIX");
5      gSuccess =
      FALSE;
    }
    else if
    (CRC!=goodCrcValue) {
10      AddMessage(
    This->fhWnd,"ERROR SEVEN");
      gSuccess =
      FALSE;
    }
15    if (!gSuccess) {
      DoNack(hComm);
      if
      (This->gVerbose) AddMessage( This->fhWnd,"NACK...");
20      This->dwNackCount++;
      if
      (This->dwNackCount==3) fDone=TRUE;
      } else fDone = TRUE;
25      This->dwInBufferIndex++;
      This->dwSubFrame=CPB_NONE;
30      This->dwFrame=CP_NOFRAME;
      break;
    }
    break;

```

```
}
```

```
}
```

```
} while (This->gReading&&!fDone);
```

```
if (fDone) {
```

5

```
// The following code is used to save the  
buffer to a file for analysis
```

```
#ifdef _DEBUG
```

```
HANDLE hCommLog;
```

10

```
DWORD dwWrite;
```

```
hCommLog=CreateFile("D:\\TEMP\\COMMLOG.BIN",GENERIC_READ | GENERIC_WRITE  
,FILE_SHARE_WRITE,NULL,CREATE_ALWAYS,0,NULL);
```

15

```
WriteFile(hCommLog,inBuffer,This->dwInBufferCount,&dwWrite,NULL);
```

```
CloseHandle(hCommLog);
```

```
#endif
```

20

```
This->gReading=FALSE;
```

```
if (This->dwNackCount < 3) {
```

```
sprintf(message,"Read %d bytes from
```

```
%s",This->dwOutBufferCount,This->gComPort);
```

```
if (This->gVerbose) AddMessage(
```

25

```
This->fhWnd, message ); // *****
```

```
strcpy((char *) inBuffer,"d=");
```

```
i2=2;
```

```
for (; i1<This->dwOutBufferCount;
```

```
i1++,i2+=2) {
```

30

```
b2=(unsigned
```

```
char)(outBuffer[i1]>>4);
```

```
inBuffer[i2]=TranslateDigitHex(b2);
```

```

inBuffer[i2+1]=TranslateDigitHex((unsigned char)(outBuffer[i1]-(b2<<4)));
    }

```

```

5          err =
PostURL(hWnd,This->gHostName,(unsigned
short)atoi(This->gHostPort),atoi(This->gUID),inBuffer,This->dwOutBufferCount
*2+2,This->gProxyName,(unsigned short)atoi(This->gProxyPort));
    }
10      else    err = 1;
        if (err==0) {
            AddMessage( This->hWnd,"Upload
Successful - please wait...");
15      NPN_GetURL(This->gInstance,This->gSourceURL,"_current");
        } else {
            AddMessage( This->hWnd,"Upload
Failed");
            Cleanup(This);
            fDone=FALSE;
20          CRC=0;
        }
        /*err = NPN_PostURL( gInstance, gURL, NULL,
dwOutBufferCount*2+2, inBuffer, FALSE);
25      if( err != NPERR_NO_ERROR ) {
        printf("Error on NPN_PostURL()");
        }*/
    }
    break;
30
case WM_PAINT: {

    hdc = BeginPaint( hWnd, &paintStruct );

```

HBRUSH hBr;

hBr = CreateSolidBrush(GetSysColor(COLOR\_WINDOW));  
GetClientRect(hWnd,&rect);

5

FillRect(hdc,&rect,hBr);

DeleteObject(hBr);

10

for (i = 0; i < gNumLines; i++ ) {

TextOut( hdc, 0, (i \* 20),

gMessageTextArray[i], strlen(gMessageTextArray[i]) );

}

15

EndPaint( hWnd, &paintStruct );

break;

}

default: {

This->fDefaultWindowProc( hWnd, Msg, wParam,

20

lParam);

}

}

return 0;

}

25

//=====